Learning rules with Adaptor Grammars
(the Berkeley edition)

Mark Johnson

joint work with Sharon Goldwater and Tom Griffiths

May, 2008

The drunk under the lamppost

Late one night, a drunk guy is crawling around under a lamppost. A cop comes up and asks him what he's doing.

"*I'm looking for my keys*," the drunk says. "*I lost them about three blocks away*."

"*So why aren't you looking for them where you dropped them?*" the cop asks.

The drunk looks at the cop, amazed that he'd ask so obvious a question. "*Because the light is better here.*"

Ideas behind talk

- Most successful statistical learning methods are *parametric*
  - PCFGs have one probability parameter per rule
  - PCFG learning: given rules and data, learn rule probabilities
- Non-parametric learning: learn parameters (rules) as well as values
- *Adaptor grammars*:
  - are a framework for specifying hierarchical nonparametric Bayesian models
  - can express a variety of linguistically-interesting structures
  - are approximated by PCFGs, where number of rules depends on data
  - attempt to put ideas behind Goldwater's models into a grammatical framework

Language acquisition as Bayesian inference

$$\underbrace{\mathsf{P(Grammar \mid Data)}}_{\text{Posterior}} \propto \underbrace{\mathsf{P(Data \mid Grammar)}}_{\text{Likelihood}} \underbrace{\mathsf{P(Grammar)}}_{\text{Prior}}$$

- Likelihood measures how well grammar describes data
- Prior expresses knowledge of grammar before data is seen
  - can be very specific (e.g., Universal Grammar)
  - can be very general (e.g., prefer shorter grammars)
- Posterior is *distribution* over grammars
  - expresses uncertainty about which grammar is correct
- But: *infinitely many* grammars may be consistent with Data

Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

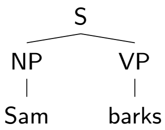
Extending Adaptor Grammars

Conclusion

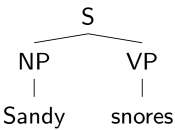
Probabilistic context-free grammars

- Rules in *Context-Free Grammars* (CFGs) expand nonterminals into sequences of terminals and nonterminals
- A *Probabilistic CFG* (PCFG) associates each nonterminal with a multinomial distribution over the rules that expand it
- Probability of a tree is the *product of the probabilities of the rules* used to construct it

Rule r	θ_r	Rule r	θ_r
S $\rightarrow$ NP VP	1.0		
NP $\rightarrow$ Sam	0.75	NP $\rightarrow$ Sandy	0.25
VP $\rightarrow$ barks	0.6	VP $\rightarrow$ snores	0.4

P (

) $= 0.45$

P (

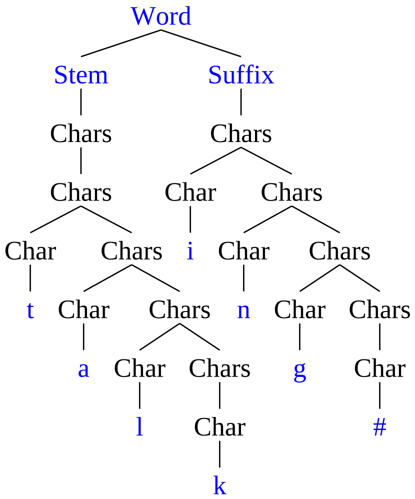
) $= 0.1$

Learning syntactic structure is hard

- Bayesian PCFG estimation works well on toy data
- Results are disappointing on “real” data
  - wrong data?
  - wrong rules?
    (rules in PCFG are given a priori; can we learn them too?)
- Strategy: study simpler cases
  - Morphological segmentation (e.g., *walking* = *walk*+*ing*)
  - Word segmentation of unsegmented utterances

A CFG for stem-suffix morphology

Word	→	Stem Suffix	Chars	→	Char
Stem	→	Chars	Chars	→	Char Chars
Suffix	→	Chars	Char	→	a | b | c | ...



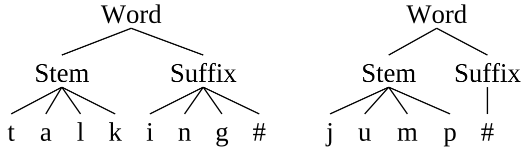
- Grammar's trees can represent any segmentation of words into stems and suffixes

⇒ Can represent true segmentation

- But grammar's *units of generalization (PCFG rules) are "too small" to learn morphemes*

A “CFG” with one rule per possible morpheme

$$
\begin{aligned}
\mathsf{Word} &\rightarrow \mathsf{Stem\ Suffix}\\
\mathsf{Stem} &\rightarrow \text{all possible stems}\\
\mathsf{Suffix} &\rightarrow \text{all possible suffixes}
\end{aligned}
$$



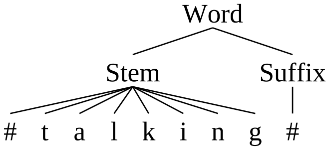
- A rule for each morpheme
  $\Rightarrow$ “PCFG” can represent probability of each morpheme
- *Unbounded number of possible rules, so this is not a PCFG*
  - not a practical problem, as only a finite set of rules could possibly be used in any particular data set

Maximum likelihood estimate for $\boldsymbol{\theta}$ is trivial

- Maximum likelihood selects $\boldsymbol{\theta}$ that minimizes KL-divergence between model and training data $\boldsymbol{W}$ distributions
- *Saturated model* in which each word is generated by its own rule replicates training data distribution $\boldsymbol{W}$ exactly

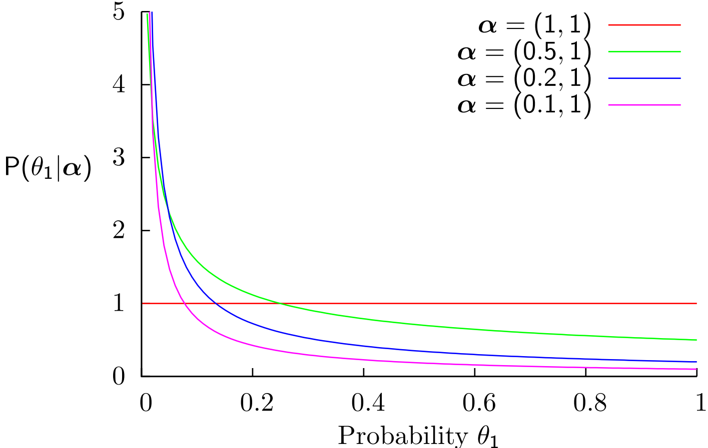
$\Rightarrow$ Saturated model is maximum likelihood estimate

- Maximum likelihood estimate does not find any suffixes



Forcing generalization via sparse Dirichlet priors

- Idea: use Bayesian prior that prefers fewer rules
- Set of rules is given a priori in Bayesian PCFGs, but can "turn rule off" by setting $\theta_{A\to\beta} \approx 0$
- Dirichlet prior with $\alpha_{A\to\beta} \approx 0$ prefers $\theta_{A\to\beta} \approx 0$



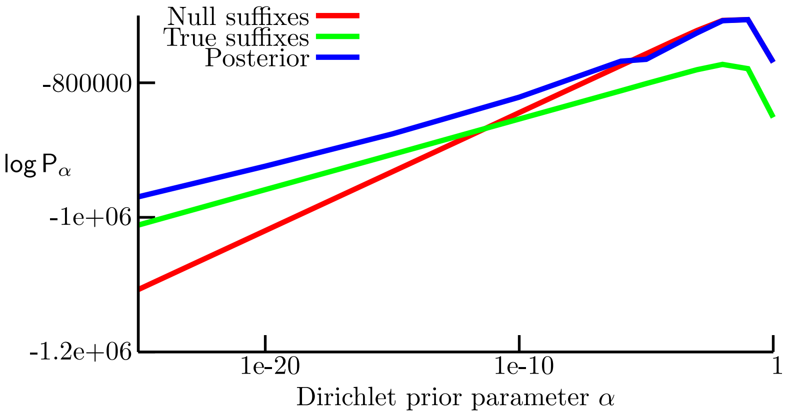
Morphological segmentation experiment

- Trained on orthographic verbs from U Penn. Wall Street Journal treebank
- Uniform Dirichlet prior prefers sparse solutions as $\alpha \rightarrow 0$
- Metropolis-within-Gibbs sampler used to sample from posterior distribution of parses
    - reanalyses each word based on a grammar estimated from the parses of the other words

Posterior samples from WSJ verb tokens

$\alpha = 0.1$		$\alpha = 10^{-5}$		$\alpha = 10^{-10}$		$\alpha = 10^{-15}$	
expect		expect		expect		expect	
expects		expects		expects		expects	
expected		expected		expected		expected	
expecting		expect	ing	expect	ing	expect	ing
include		include		include		include	
includes		includes		includ	es	includ	es
included		included		includ	ed	includ	ed
including		including		including		including	
add		add		add		add	
adds		adds		adds		add	s
added		added		add	ed	added	
adding		adding		add	ing	add	ing
continue		continue		continue		continue	
continues		continues		continue	s	continue	s
continued		continued		continu	ed	continu	ed
continuing		continuing		continu	ing	continu	ing
report		report		report		report	

Log posterior of models on token data



- Correct solution is nowhere near as likely as posterior

$\Rightarrow$ model is wrong!

Independence assumptions in PCFGs

- Context-free grammars are "context-free" because the possible expansions of each node do not depend on expansions of other nodes
- Probabilistic CFGs extend this by requiring each node expansion to be *statistically independent* (conditioned on the node's label)
- This is a very strong assumption, which is often false!
- Morphology grammar contains rule:

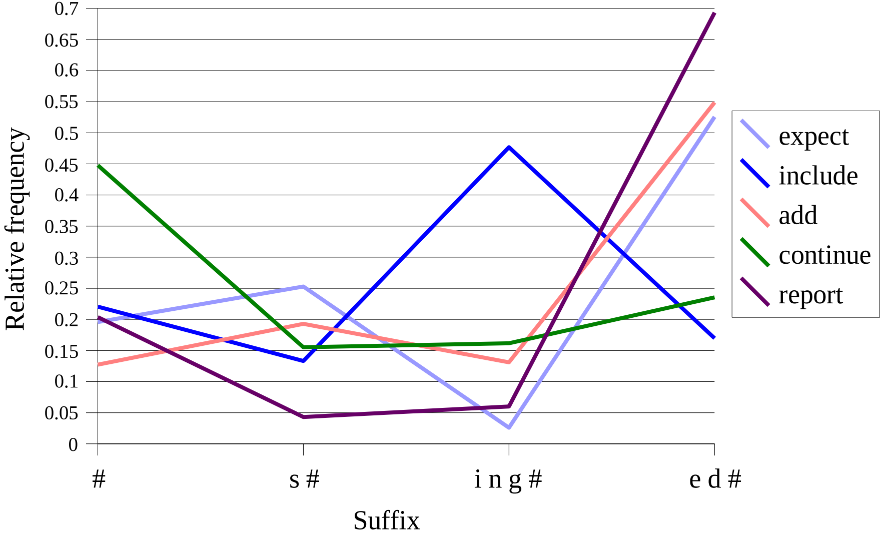
$$\mathsf{Word} \rightarrow \mathsf{Stem}\ \mathsf{Suffix}$$

- Corresponding independence assumption:

$$\mathsf{P(Word)} = \mathsf{P(Stem)}\,\mathsf{P(Suffix)}$$

causes PCFG model of morphology to fail

Relative frequencies of inflected verb forms



Types and tokens

- A word *type* is a distinct word shape
- A word *token* is an occurrence of a word

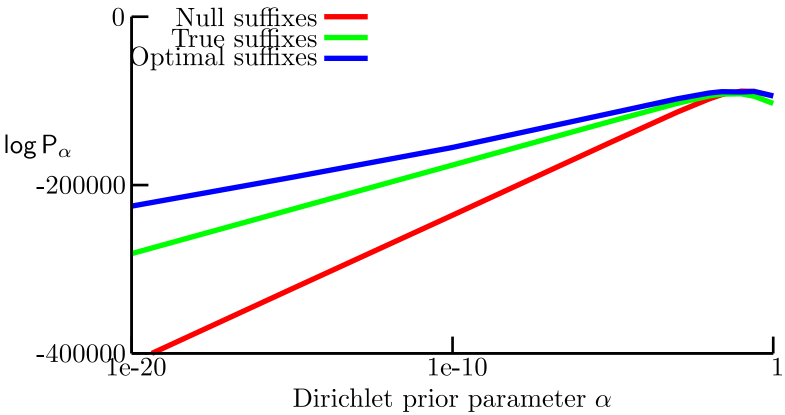
$$\begin{aligned} \mathsf{Data} &= \text{“the cat chased the other cat”} \\ \mathsf{Tokens} &= \text{“the”, “cat”, “chased”, “the”, “other”, “cat”} \\ \mathsf{Types} &= \text{“the”, “cat”, “chased”, “other”} \end{aligned}$$

- Estimating θ from *word types* rather than word tokens eliminates (most) frequency variation
  - 4 common verb suffixes, so when estimating from verb types $\theta_{\mathsf{Suffix} \to \mathtt{ing\#}} \approx 0.25$
- Several psycholinguists believe that humans learn morphology from word types
- Goldwater et al investigated a morphology-learning model that learnt from an interpolation of types and tokens

Posterior samples from WSJ verb *types*

$\alpha = 0.1$		$\alpha = 10^{-5}$		$\alpha = 10^{-10}$		$\alpha = 10^{-15}$	
expect		expect		expect		exp	ect
expects		expect	s	expect	s	exp	ects
expected		expect	ed	expect	ed	exp	ected
expect	ing	expect	ing	expect	ing	exp	ecting
include		includ	e	includ	e	includ	e
include	s	includ	es	includ	es	includ	es
included		includ	ed	includ	ed	includ	ed
including		includ	ing	includ	ing	includ	ing
add		add		add		add	
adds		add	s	add	s	add	s
add	ed	add	ed	add	ed	add	ed
adding		add	ing	add	ing	add	ing
continue		continu	e	continu	e	continu	e
continue	s	continu	es	continu	es	continu	es
continu	ed	continu	ed	continu	ed	continu	ed
continuing		continu	ing	continu	ing	continu	ing
report		report		repo	rt	rep	ort

Log posterior of models on type data



- Correct solution is close to optimal at $\alpha = 10^{-3}$

Desiderata for an extension of PCFGs

- PCFG rules are "too small" to be effective units of generalization
  $\Rightarrow$ generalize over groups of rules
  $\Rightarrow$ units of generalization should be chosen based on data
- Type-based inference mitigates non-context-free dependencies
  $\Rightarrow$ Hierarchical Bayesian model where:
    - context-free rules generate types
    - another process replicates types to produce tokens
- *Adaptor grammars:*
    - learn probability of entire subtrees (how a nonterminal expands to terminals)
    - use grammatical hierarchy to define a Bayesian hierarchy, from which type-based inference emerges
    - inspired by Goldwater's work

Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion

Dirichlet-Multinomials with many outcomes

- Dirichlet prior $\boldsymbol{\alpha}$, observed data $\boldsymbol{X} = (X_1, \ldots, X_n)$

$$\mathsf{P}(X_{n+1} = k \mid \boldsymbol{X}, \boldsymbol{\alpha}) \;\; \propto \;\; \alpha_k + n_k(\boldsymbol{X})$$

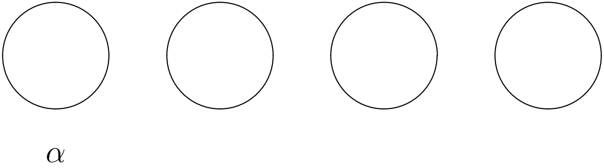
- Consider a sequence of Dirichlet-multinomials where:
  - total Dirichlet pseudocount is fixed $\alpha_\bullet = \sum_{k=1}^{m} \alpha_k$, and
  - prior uniform over outcomes $1, \ldots, m$, so $\alpha_k = \alpha_\bullet / m$
  - number of outcomes $m \to \infty$

$$\mathsf{P}(X_{n+1} = k \mid \boldsymbol{X}, \alpha_\bullet) \;\; \propto \;\; \begin{cases} n_k(\boldsymbol{X}) & \text{if } n_k(\boldsymbol{X}) > 0 \\ \\ \alpha_\bullet / m & \text{if } n_k(\boldsymbol{X}) = 0 \end{cases}$$

But when $m \gg n$, most k are unoccupied (i.e., $n_k(\boldsymbol{X}) = 0$)

$\Rightarrow$ *Probability of a previously seen outcome* $k \;\; \propto \;\; n_k(\boldsymbol{X})$
Probability of an outcome never seen before $\;\; \propto \;\; \alpha_\bullet$

Labeled Chinese restaurant processes (1a)

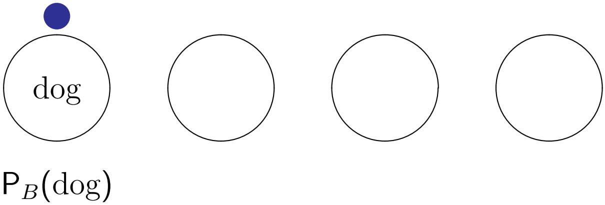


Generated sequence:

- Each occupied "table" has a label (a "dish"), sampled from P_B
- Customer $n + 1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto\ n_k$
  - sits at new table $k = m + 1$ with probability $\propto\ \alpha$
- Emit label (dish) on table
  - if table doesn't have a label, generate one from P_B

$\Rightarrow$ only pay probability cost for label *once per table*

Labeled Chinese restaurant processes (1b)

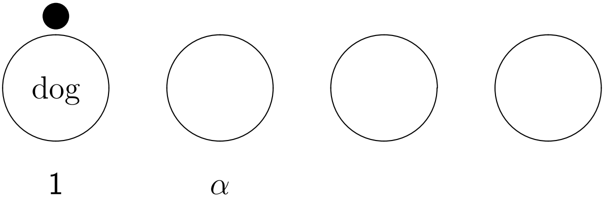


Generated sequence: dog

- Each occupied "table" has a label (a "dish"), sampled from P_B
- Customer $n+1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto \; n_k$
  - sits at new table $k = m+1$ with probability $\propto \; \alpha$
- Emit label (dish) on table
  - if table doesn't have a label, generate one from P_B

$\Rightarrow$ only pay probability cost for label *once per table*

Labeled Chinese restaurant processes (2a)

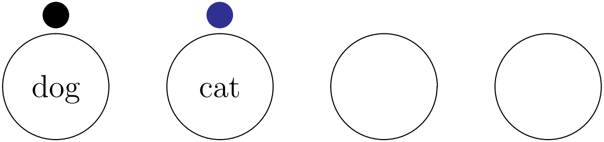


Generated sequence: dog

- Each occupied "table" has a label (a "dish"), sampled from P_B
- Customer $n + 1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto\ n_k$
  - sits at new table $k = m + 1$ with probability $\propto\ \alpha$
- Emit label (dish) on table
  - if table doesn't have a label, generate one from P_B

$\Rightarrow$ only pay probability cost for label *once per table*

Labeled Chinese restaurant processes (2b)



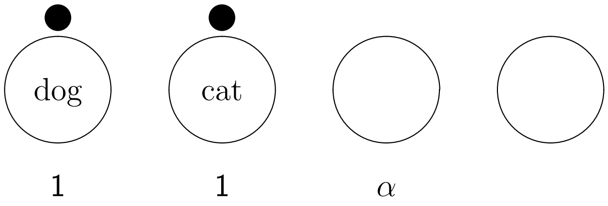
$\mathsf{P}_B(\text{cat})$

Generated sequence: dog, cat

- Each occupied "table" has a label (a "dish"), sampled from P_B
- Customer $n+1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto \; n_k$
  - sits at new table $k = m+1$ with probability $\propto \; \alpha$
- Emit label (dish) on table
  - if table doesn't have a label, generate one from P_B

$\Rightarrow$ only pay probability cost for label *once per table*

Labeled Chinese restaurant processes (3a)

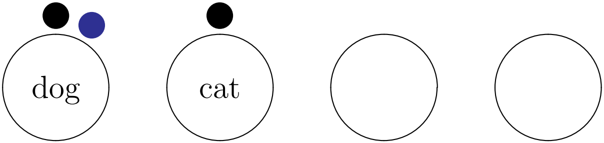


Generated sequence: dog, cat

- Each occupied "table" has a label (a "dish"), sampled from P_B
- Customer $n+1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto\ n_k$
  - sits at new table $k = m+1$ with probability $\propto\ \alpha$
- Emit label (dish) on table
  - if table doesn't have a label, generate one from P_B

  $\Rightarrow$ only pay probability cost for label *once per table*

Labeled Chinese restaurant processes (3b)



Generated sequence: dog, cat, dog

- Each occupied "table" has a label (a "dish"), sampled from P_B
- Customer $n+1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto\ n_k$
  - sits at new table $k = m+1$ with probability $\propto\ \alpha$
- Emit label (dish) on table
  - if table doesn't have a label, generate one from P_B

  $\Rightarrow$ only pay probability cost for label *once per table*

From Chinese restaurants to Dirichlet processes

- Chinese restaurant processes map a distribution P_B to a stream of samples from a different distribution with the same support
- CRPs specify the *conditional distribution* of the next outcome given the previous ones
- Each CRP run can produce a different distribution over labels
- It defines a mapping from α and P_B to a *distribution over distributions* $\mathsf{DP}(\alpha, \mathsf{P}_B)$
- $\mathsf{DP}(\alpha, \mathsf{P}_B)$ is called a *Dirichlet process* (DP) with *concentration parameter* α and *base distribution* P_B
- The base distribution P_B can be defined by a DP $\Rightarrow$ *hierarchy* of DPs

Nonparametric extensions of PCFGs

- Chinese restaurant processes are a nonparametric extension of Dirichlet-multinomials because the number of states (occupied tables) depends on the data
- Two obvious nonparametric extensions of PCFGs:
  - let the number of nonterminals grow unboundedly
    - refine the nonterminals of an original grammar e.g., $\mathsf{S}_{35} \rightarrow \mathsf{NP}_{27}\ \mathsf{VP}_{17}$

    $\Rightarrow$ infinite PCFG
  - let the number of rules grow unboundedly
    - "new" rules are compositions of several rules from original grammar
    - equivalent to caching tree fragments

    $\Rightarrow$ adaptor grammars
- No reason both can't be done together ...

Outline

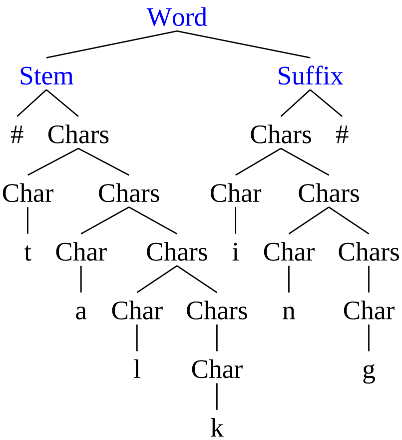
Adaptor grammars: informal description

- An adaptor grammar has a set of CFG rules
- These determine the possible structures as in a CFG
- A subset of the nonterminals are *adapted*
- *Unadapted nonterminals* expand by picking a rule and recursively expanding its children, as in a PCFG
- *Adapted nonterminals* can expand in two ways:
  - by picking a rule and recursively expanding its children, or
  - by generating a previously generated tree (with probability proportional to the number of times previously generated)
- Each adapted subtree behaves like a new rule added to the grammar
- The CFG rules of the adapted nonterminals determine the *base distribution* over these trees

Adaptor grammars as generative processes

- The sequence of trees generated by an adaptor grammar are *not* independent
  - it *learns* from the trees it generates
  - if an adapted subtree has been used frequently in the past, it's more likely to be used again
- (but the sequence of trees is *exchangable*)
- An *unadapted nonterminal* A expands using $A \rightarrow \beta$ with probability $\theta_{A \rightarrow \beta}$
- An *adapted nonterminal* A expands:
  - to a subtree τ rooted in A with probability proportional to the number of times τ was previously generated
  - using $A \rightarrow \beta$ with probability proportional to $\alpha_A \theta_{A \rightarrow \beta}$

Adaptor grammar morphology example



Word	$\rightarrow$	Stem Suffix
Stem	$\rightarrow$	# Chars
Suffix	$\rightarrow$	#
Suffix	$\rightarrow$	Chars #
Chars	$\rightarrow$	Char
Chars	$\rightarrow$	Char Chars
Char	$\rightarrow$	a | ... | z

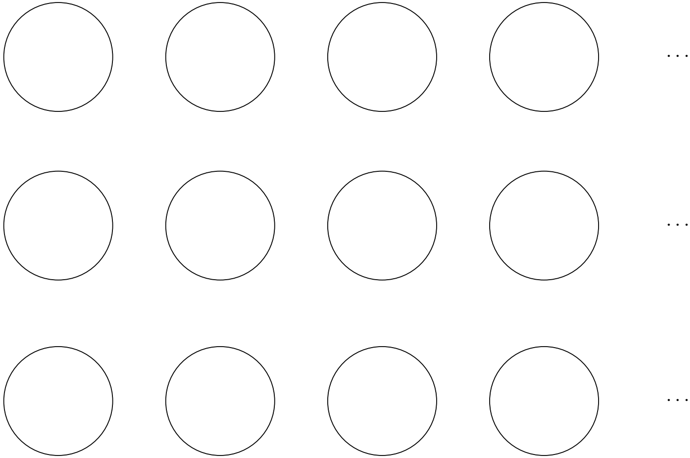
- Stem and Suffix rules generate all possible stems and suffixes
- Adapt Word, Stem and Suffix nonterminals
- One Chinese Restaurant process per adapted nonterminal

Morphology adaptor grammar (0)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

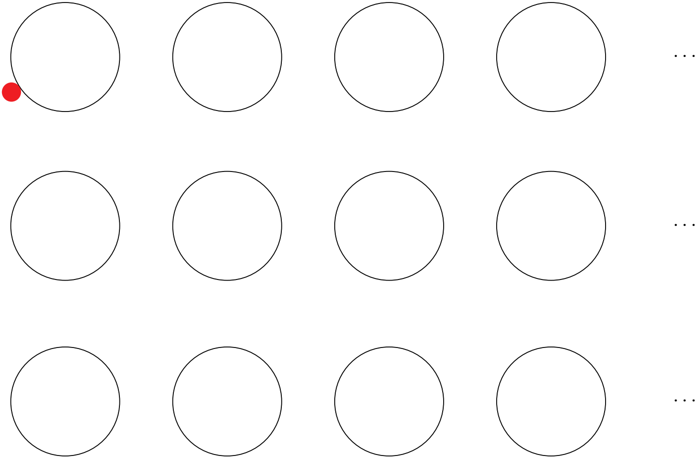
Morphology adaptor grammar (1a)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

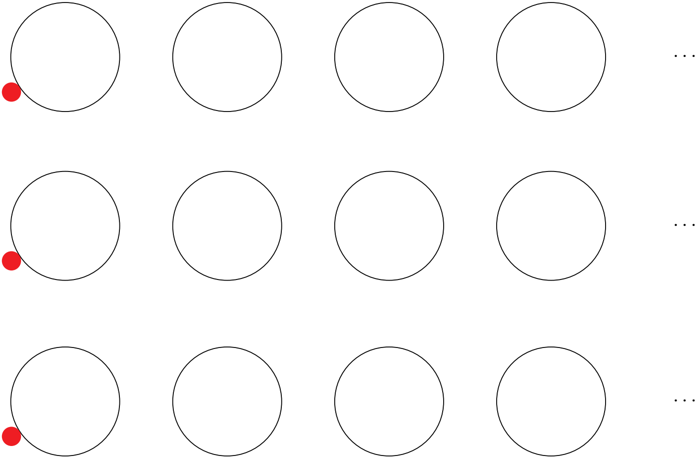


Morphology adaptor grammar (1b)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



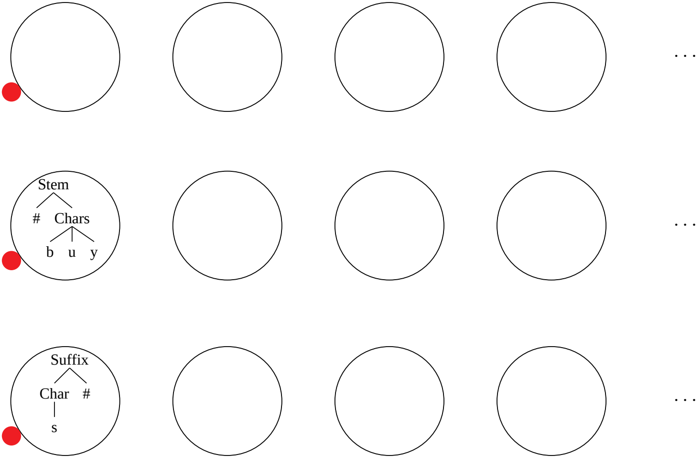
Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

Morphology adaptor grammar (1c)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



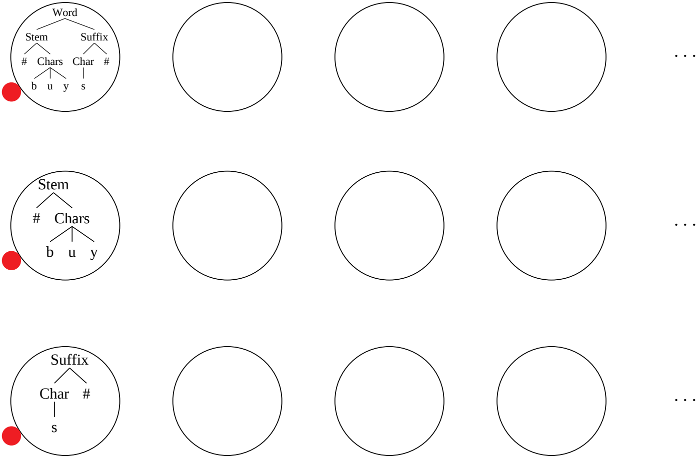
Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

Morphology adaptor grammar (1d)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



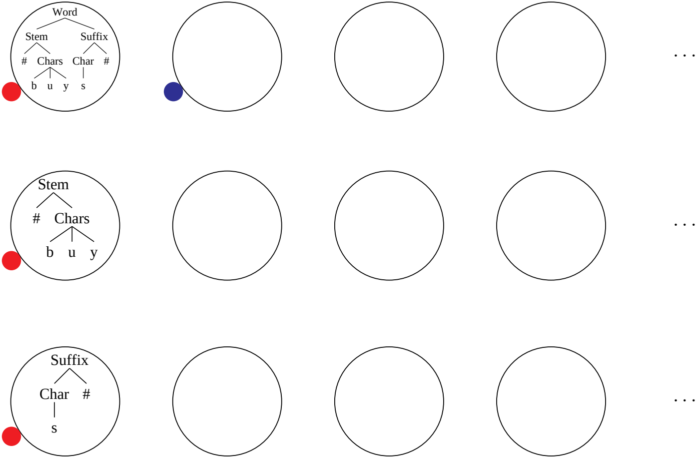
Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

Morphology adaptor grammar (2a)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

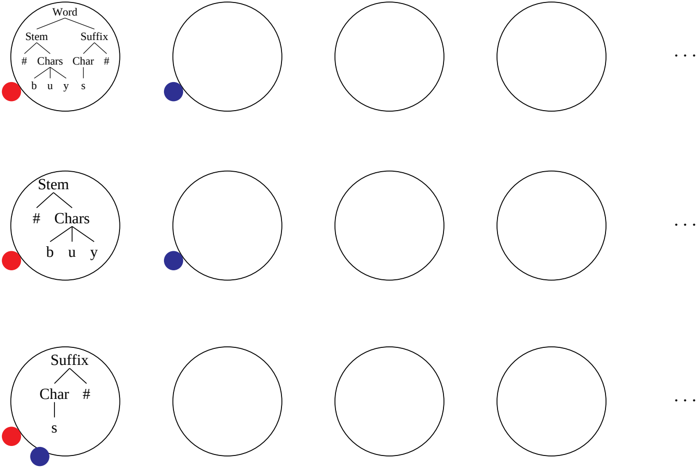
Morphology adaptor grammar (2b)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

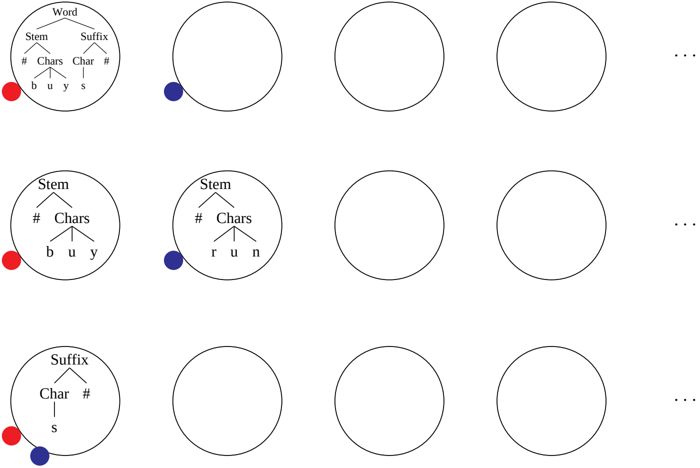


Morphology adaptor grammar (2c)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

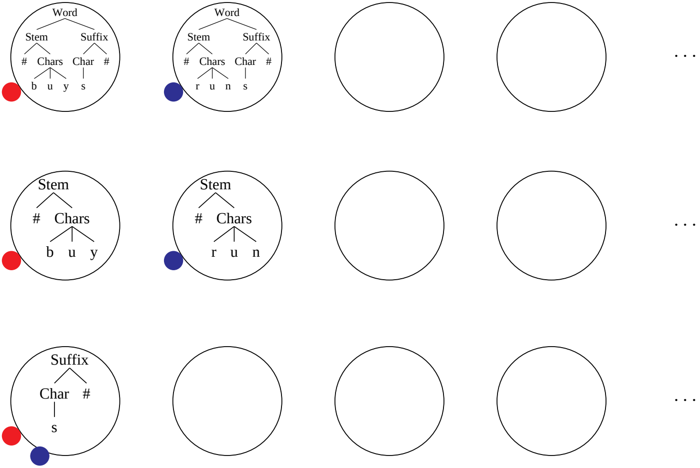
Morphology adaptor grammar (2d)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z



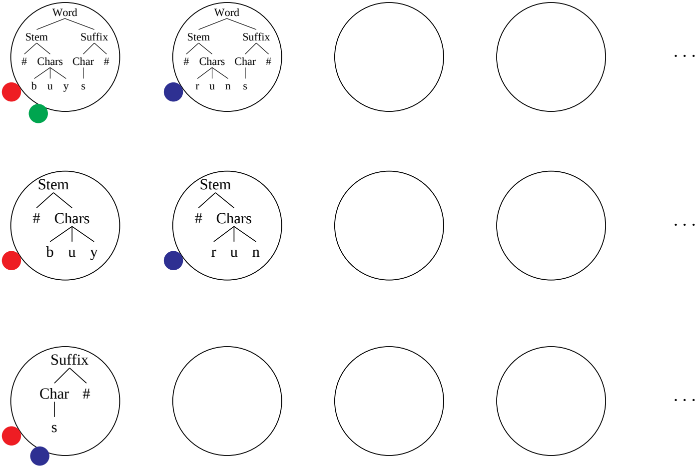
Morphology adaptor grammar (3)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

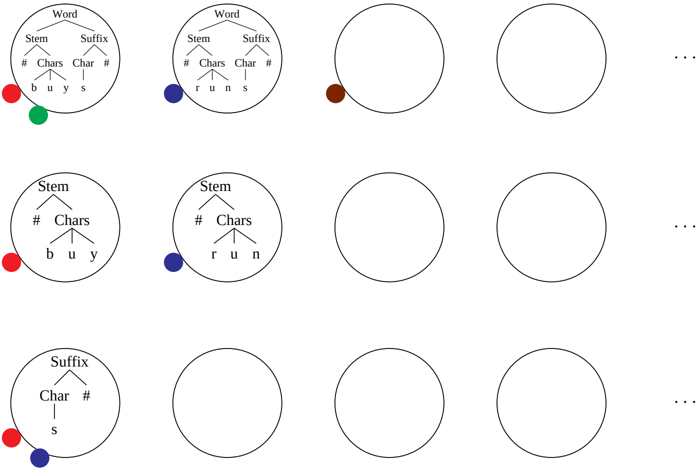


Morphology adaptor grammar (4a)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



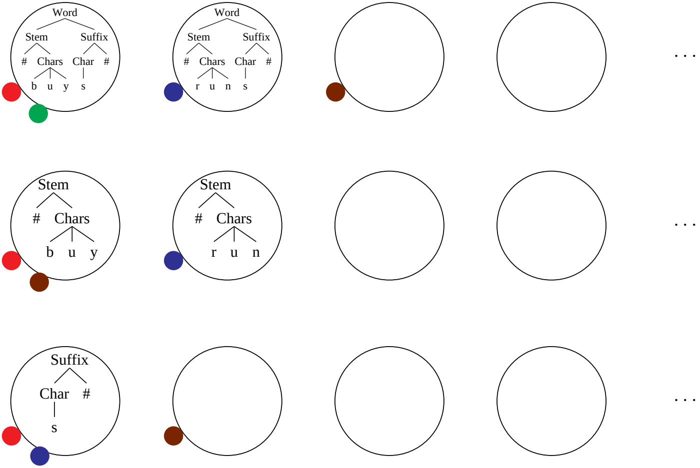
Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

Morphology adaptor grammar (4b)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #



Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z

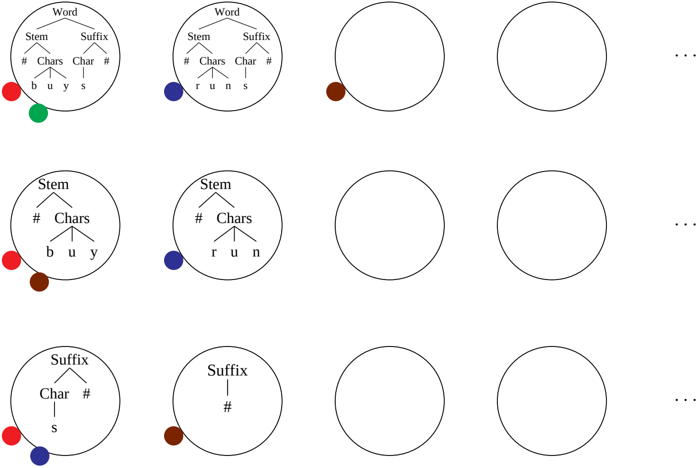
Morphology adaptor grammar (4c)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z



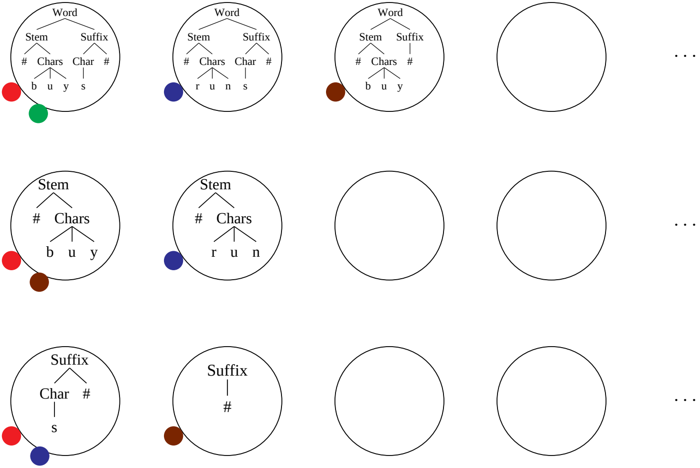
Morphology adaptor grammar (4d)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a . . . z



Properties of adaptor grammars

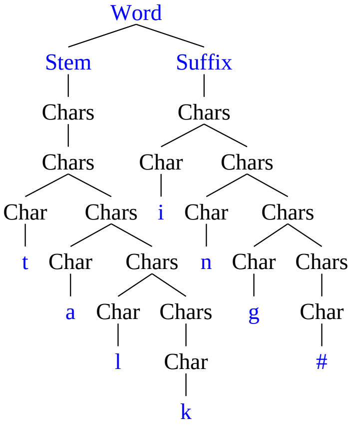
- Possible trees generated by CFG rules
  but the probability of each adapted tree is estimated separately
- Probability of a subtree τ is proportional to:
  - the number of times τ was seen before
    $\Rightarrow$ "rich get richer" dynamics (Zipf distributions)
  - plus α_A times prob. of generating it via PCFG expansion

$\Rightarrow$ Useful compound structures can be *more probable than their parts*

- PCFG rule probabilities estimated *from table labels*
  $\Rightarrow$ learns from types, not tokens
  $\Rightarrow$ dampens frequency variation

Bayesian hierarchy inverts grammatical hierarchy

- Grammatically, a Word is composed of a Stem and a Suffix, which are composed of Chars
- To generate a new Word from an adaptor grammar
  - reuse an old Word, or
  - generate a fresh one from the base distribution, i.e., generate a Stem and a Suffix
- Lower in the tree
  ⇒ higher in Bayesian hierarchy



Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

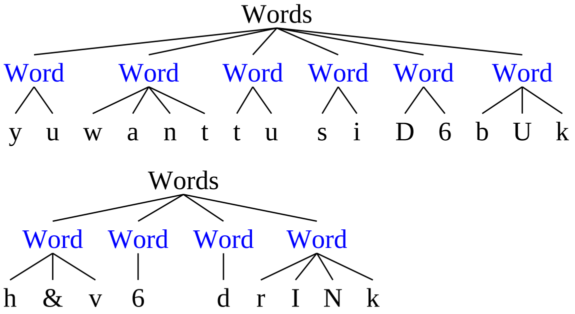
Extending Adaptor Grammars

Conclusion

Unigram model of word segmentation

- Unigram model: each word is generated independently
- Input is *unsegmented broad phonemic transcription* (Brent)
  Example: y u w a n t t u s i D 6 b u k
- Adaptor for Word non-terminal caches previously seen words

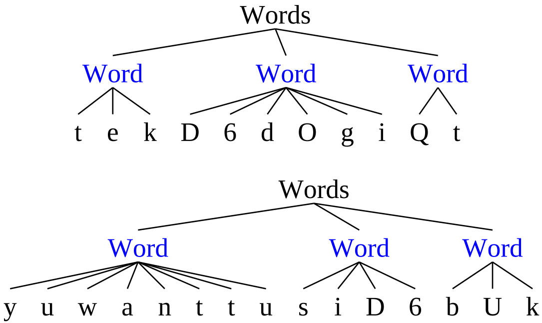
Words → Word$^+$
Word → Phoneme$^+$



- Unigram word segmentation on Brent corpus: 55% token f-score

Unigram model often finds collocations

- Unigram word segmentation model assumes each word is generated independently
- But there are strong inter-word dependencies (collocations)
- Unigram model can only capture such dependencies by analyzing collocations as words (Goldwater 2006)



Unigram word segmentation grammar learnt

- Based on the base grammar rules

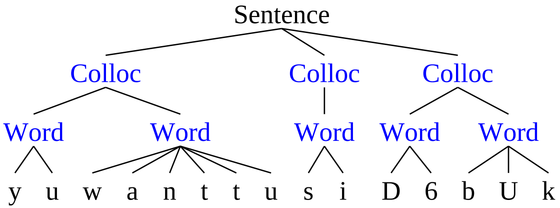
$$\text{Words} \to \text{Word}^+$$
$$\text{Word} \to \text{Phoneme}^+$$

the adapted grammar contains 1,712 rules such as:

15758	Words → Word Words
9791	Words → Word
1660	Word → Phoneme$^+$
402	Word → y u
137	Word → I n
111	Word → w I T
100	Word → D 6 d O g i
45	Word → I n D 6
20	Word → I n D 6 h Q s

Modeling collocations improves segmentation

Sentence → Colloc$^+$
Colloc → Word$^+$
Word → Phoneme*

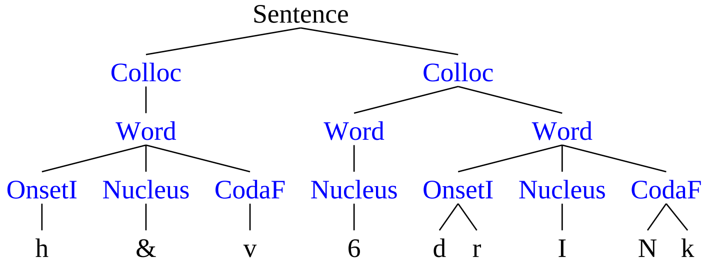


- A Colloc(ation) consists of one or more words
- Both Words and Collocs are adapted (learnt)
- Significantly improves word segmentation accuracy over unigram model (75% f-score; ≈ Goldwater's bigram model)
- Two levels of Collocations improves slightly (76%)

Syllables + Collocations + Word segmentation

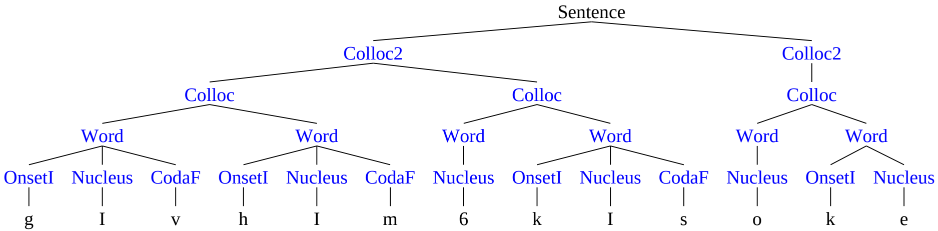
Sentence $\rightarrow$ Colloc$^+$
Word $\rightarrow$ SyllableIF
Word $\rightarrow$ SyllableI Syllable SyllableF
Onset $\rightarrow$ Consonant$^+$
Nucleus $\rightarrow$ Vowel$^+$

Colloc $\rightarrow$ Word$^+$
Word $\rightarrow$ SyllableI SyllableF
Syllable $\rightarrow$ (Onset) Rhyme
Rhyme $\rightarrow$ Nucleus (Coda)
Coda $\rightarrow$ Consonant$^+$



- With no supra-word generalizations, f-score = 68%
- With 2 Collocation levels, f-score = 84%
- Without distinguishing initial/final clusters, f-score = 82%

Syllables + 2-level Collocations + Word segmentation



Word segmentation results summary

		Collocation levels above the word			
		none	1 level	2 levels	3 levels
Below the word	none	0.55	0.73	0.75	0.74
	morphemes	0.35	0.55	0.79	0.78
	syllables	0.32	0.69	0.82	0.81
	syllables IF	0.46	0.68	**0.84**	**0.84**

- We can learn collocations and syllable structure together with word segmentation, even though we don't know where the word boundaries are
- Learning these together improves word segmentation accuracy
  - are there other examples of *synergistic interaction* in language learning?

Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion

Estimating adaptor grammars

- Need to estimate:
  - cached subtrees τ for adapted nonterminals
  - (optional) DP parameters $\boldsymbol{\alpha}$ for adapted nonterminals
  - (optional) probabilities $\boldsymbol{\theta}$ of base grammar rules
- Component-wise Metropolis-within-Gibbs sampler
  - components are parse tree T_i for each string W_i
  - sample T_i from $\mathsf{P}(T|W_i, \boldsymbol{T}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\theta})$ for each sentence W_i in turn
- Sampling directly from conditional distribution of parses seems intractable
  - construct PCFG proposal grammar $G'(\boldsymbol{T}_{-i})$ on the fly
  - each table label τ corresponds to a production in PCFG approximation
  - Use accept/reject to convert samples from PCFG approx to samples from adaptor grammar

Metropolis-with-Gibbs sampler

- Collapsed Gibbs sampler: resample parse T_i given W_i and $\boldsymbol{T}_{-i}$
- Table counts change within a parse tree
  - $\Rightarrow$ grammar is not context-free
  - $\Rightarrow$ breaks standard dynamic programming
  - $\Rightarrow$ Metropolis accept/reject for each Gibbs sample
- PCFG can express probability of selecting a table given $\boldsymbol{T}_{-i}$
  - ignores changing table counts within single parse
- Rules of PCFG proposal grammar G' consist of:
  - rules $A \to \beta$ from base PCFG: $\theta'_{A\to\beta} \;\propto\; \alpha_A \theta_{A\to\beta}$
  - A rule $A \to \textsc{Yield}(\tau)$ for each table τ in A's restaurant: $\theta'_{A\to\textsc{Yield}(\tau)} \;\propto\; n_\tau$, the number of customers at table τ
- Parses of G' can be mapped back to adaptor grammar parses

Bayesian priors on adaptor grammar parameters

- Parameters of adaptor grammars:
  - probabilities $\theta_{A \to \beta}$ of base grammar rules $A \to \beta$
  - concentration parameters α_A of adapted nonterminals A
- Put Bayesian priors on these parameters
  - (Uniform) Dirichlet prior on base grammar rule probabilities $\boldsymbol{\theta}$
  - Vague Gamma prior on concentration parameter on α_A
- We also use a generalization of CRPs called "Pitman-Yor processes", and put a uniform Dirichlet prior on its a parameter
- We use a Metropolis-Hastings sampler for a and b parameters
  - a is sampled from sequence of increasingly narrow Dirichlets
  - b is sampled from sequence of increasingly narrow Gammas
- Seems to improve performance with complicated grammars

Random initialization is better than incremental initialization

- Incremental initialization: assign parse for W_i based on $\boldsymbol{T}_{1,i-1}$
- Random initialization: initially assign parses T_i *randomly*
- Incremental initialization seems to get stuck in local optima

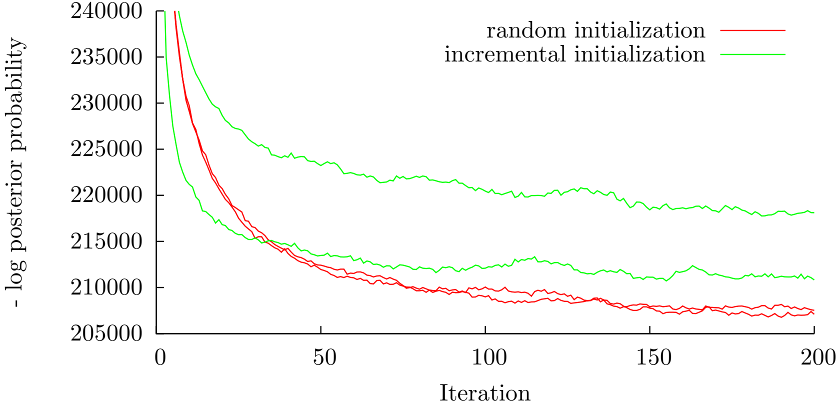


Table label resampling improves mobility

- Gibbs algorithm: resample T_i given W_i and $\boldsymbol{T}_{-i}$
- Table label resampling resamples the labels on each table
  - can change parses for many sentences at once

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char

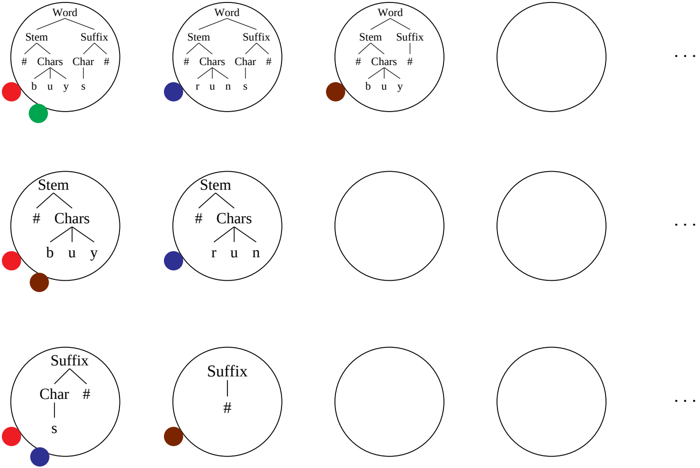
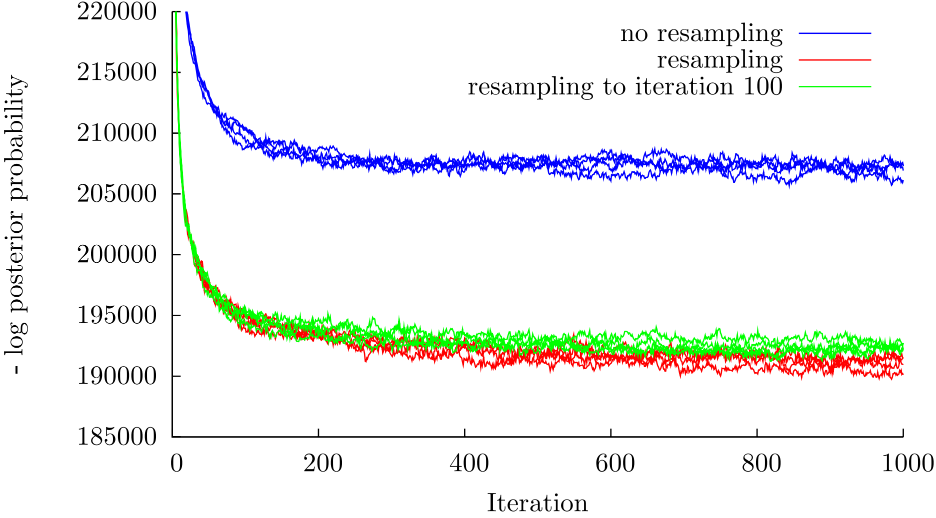
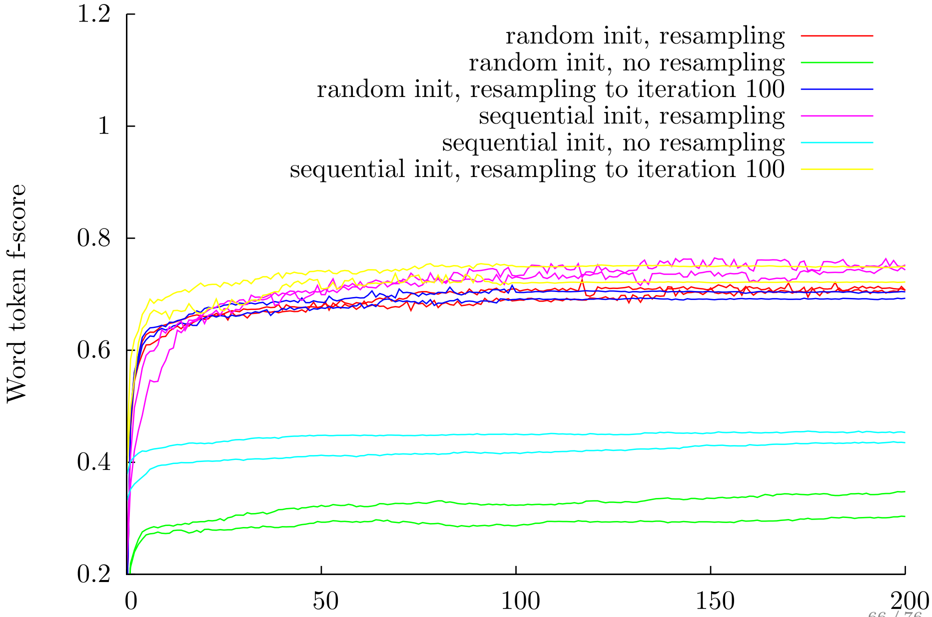


Table label resampling with Colloc grammar



Segmentation accuracy with Colloc grammar



Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion

Issues with adaptor grammars

- Recursion *through adapted nonterminals* seems problematic
  - New tables are created as each node is encountered top-down
  - But the tree labeling the table is only known after the whole subtree has been completely generated
  - If adapted nonterminals are recursive, might pick a table whose label we are currently constructing. What then?
- Extend adaptor grammars so adapted fragments can end at nonterminals a la DOP (currently always go to terminals)
  - Adding "exit probabilities" to each adapted nonterminal
  - In some approaches, fragments can grow "above" existing fragments, but can't grow "below" (O'Donnell)
- Adaptor grammars *conflate grammatical and Bayesian hierarchies*
  - Might be useful to disentangle them with *meta-grammars*

Context-free grammars

A *context-free grammar* (CFG) consists of:

- a finite set N of *nonterminals*,
- a finite set W of *terminals* disjoint from N,
- a finite set R of *rules* $A \to \beta$, where $A \in N$ and $\beta \in (N \cup W)^\star$
- a *start symbol* $S \in N$.

Each $A \in N \cup W$ *generates* a set $\mathcal{T}_A$ of trees.

These are the smallest sets satisfying:

- If $A \in W$ then $\mathcal{T}_A = \{A\}$.
- If $A \in N$ then:

$$\mathcal{T}_A = \bigcup_{A \to B_1 \ldots B_n \in R_A} \textsc{Tree}_A(\mathcal{T}_{B_1}, \ldots, \mathcal{T}_{B_n})$$

where $R_A = \{A \to \beta : A \to \beta \in R\}$, and

$$\textsc{Tree}_A(\mathcal{T}_{B_1}, \ldots, \mathcal{T}_{B_n}) = \left\{ \begin{array}{c} A \\ \overbrace{t_1 \ \ldots \ t_n} \end{array} : \begin{array}{l} t_i \in \mathcal{T}_{B_i}, \\ i = 1, \ldots, n \end{array} \right\}$$

The set of trees generated by a CFG is $\mathcal{T}_S$.

Probabilistic context-free grammars

A *probabilistic context-free grammar* (PCFG) is a CFG and a vector $\boldsymbol{\theta}$, where:

- $\theta_{A \to \beta}$ is the probability of expanding the nonterminal A using the production $A \to \beta$.

It defines distributions G_A over trees $\mathcal{T}_A$ for $A \in N \cup W$:

$$G_A = \begin{cases} \delta_A & \text{if } A \in W \\ \displaystyle\sum_{A \to B_1 \dots B_n \in R_A} \theta_{A \to B_1 \dots B_n} \mathrm{TD}_A(G_{B_1}, \dots, G_{B_n}) & \text{if } A \in N \end{cases}$$

where δ_A puts all its mass onto the singleton tree A, and:

$$\mathrm{TD}_A(G_1, \dots, G_n) \left(\begin{array}{c} A \\ \overline{t_1 \ \dots \ t_n} \end{array} \right) = \prod_{i=1}^{n} G_i(t_i).$$

$\mathrm{TD}_A(G_1, \dots, G_n)$ is a distribution over $\mathcal{T}_A$ where each subtree t_i is generated independently from G_i.

DP adaptor grammars

An adaptor grammar $(G, \boldsymbol{\theta}, \boldsymbol{\alpha})$ is a PCFG $(G, \boldsymbol{\theta})$ together with a parameter vector $\boldsymbol{\alpha}$ where for each $A \in N$, α_A is the parameter of the Dirichlet process associated with A.

$$\begin{aligned} G_A &\sim \mathsf{DP}(\alpha_A, H_A) \text{ if } \alpha_A > 0 \\ &= H_A \qquad\qquad\quad \text{if } \alpha_A = 0 \end{aligned}$$

$$H_A = \sum_{A \to B_1 \ldots B_n \in R_A} \theta_{A \to B_1 \ldots B_n} \mathrm{TD}_A(G_{B_1}, \ldots, G_{B_n})$$

The grammar generates the distribution G_S.
One Dirichlet Process for each adapted non-terminal A (i.e., $\alpha_A > 0$).

Recursion in adaptor grammars

- The probability of joint distributions $(\boldsymbol{G}, \boldsymbol{H})$ is defined by:

$$
\begin{aligned}
G_A &\sim \mathsf{DP}(\alpha_A, H_A) && \text{if } \alpha_A > \mathsf{0} \\
&= H_A && \text{if } \alpha_A = \mathsf{0}
\end{aligned}
$$

$$
H_A = \sum_{A \to B_1 \ldots B_n \in R_A} \theta_{A \to B_1 \ldots B_n} \mathrm{TD}_A(G_{B_1}, \ldots, G_{B_n})
$$

- This holds *even if adaptor grammar is recursive*
- Question: when does this define a *distribution* over $(\boldsymbol{G}, \boldsymbol{H})$?

Adaptive fragment grammars

- Disentangle syntactic and Bayesian hierarchy
  - *Adaptive metagrammar* generates *fragment distributions*
  - which plug together as in tree substitution grammar
- Tree fragment sets $\mathcal{P}_A, A \in N$ are smallest sets satisfying:

$$\mathcal{P}_A = \bigcup_{A \to B_1 \ldots B_n \in R_A} \textsc{Tree}_A(\{B_1\} \cup \mathcal{P}_{B_1}, \ldots, \{B_n\} \cup \mathcal{P}_{B_n})$$

- Grammar's distributions G_A over $\mathcal{T}_A$ defined using *fragment distributions* F_A over $\mathcal{P}_A$ (*generalized PCFG rules*)

$$G_A = \sum_{\overset{A}{\triangle}_{B_1 \ldots B_n} \in \mathcal{P}_A} F_A\left(\overset{A}{\triangle}_{B_1 \ldots B_n}\right) \ \mathrm{TD}_{\overset{A}{\triangle}_{B_1 \ldots B_n}}(G_{B_1}, \ldots, G_{B_n})$$

- A fragment grammar generates the distribution G_S

Adaptive fragment distributions

- H_A is a PCFG distribution over $\mathcal{P}_A$

$$H_A \quad = \sum_{A \to B_1 \ldots B_n \in R_A} \theta_{A \to B_1 \ldots B_n} \mathrm{TD}_A(\eta\, \delta_{B_1} + (1 - \eta) H_{B_1}, \ldots)$$

  where η is the *fragment exit probability*
- Obtain F_A by *adapting the H_A distribution*

$$F_A \quad \sim \quad \mathsf{DP}(\alpha_A, H_A)$$

- This construction can be *iterated*, i.e., replace $\boldsymbol{\theta}$ with another fragment distribution
- Question: if we iterate this, when does the *fixed point* exist, and what is it?

Outline

Summary and future work

- Adaptor grammars “adapt” their distribution to the strings they have generated
- They learn the probabilities of the subtrees of the adapted nonterminals
- This makes adaptor grammars *non-parametric*; the subtrees they cache depends on the data
- A variety of different linguistic phenomena can be described with adaptor grammars
- Because they are grammars, they are easy to design and compose
- The basic approach seems quite flexible
  - many possible extensions of Adaptor Grammars
- MCMC sampling algorithm may not scale well to large data or complicated grammars. Are there better estimators?